Unit 2

- [ ] [Chatgpt link](https://chatgpt.com/share/66efee7e-5a90-800f-a150-681efc0b6a91)

TCP

- Transmission control protocol
- Principles of reliable data transfer
  - Packet loss reasons
    - Corrupt packet dropped by reciever
    - Packet drop at router due to lack of buffer space
    - Long queuing delays in the router
  - Reliable data transfer
    - Requires connection oriented approach
    - Throughput and delay may be compromised
- Predecessors
  - RDT protocol
    - Stop and wait RDT protocol
      - RDT 1.0
        - Assumes no bit errors and no packet delays
        - Solution: Works on the principle of acknowledgement
        - Next packet is sent after the acknowledgment
      - RDT 2.0
        - Bit errors can occure only in the packet transmitted
        - No packet delays assumed
        - Solution: Introduction of checksum
        - If checksum corrupt, send back NAK packet
        - Else send back ACK
      - RDT 2.2
        - Bit error can occur both ways
        - Assume no packet delays
        - Difficult to distinguish between old and new packets
        - ACK and NAK can also get corrupted
        - Solution: Sequence numbers (use 0 and 1 alternatively) and checksum
        - Host B just uses ACK packets with sequence (0 or 1)
      - RDT 4.0
        - Bit errors and packet delays occur in 2 way packet transmission
        - Solution: Host A starts a timer as soon as packet is transmitted

        - If ACK packet from host B is lost/old/corrupt host host A retransmits after timer expires
        - Else host A sends next packet and restart timer
    - Pipelining RDT protocols - better link utilization
        - GBN
        - SR
- TCP
    - Hybrid of above RDT protocol with some additions
    - More robust
    - Adaptive to network congestion

Pipelining in Data Transfer: Simple Overview

When data is sent over a network, it often gets split into smaller parts called packets. Normally, the sender waits for the receiver to confirm that each packet was received correctly before sending the next one. This is called a Stop-and-Wait approach. However, this can be slow because the sender has to pause and wait for an acknowledgment (ACK) after every packet.

To speed things up, we use pipelining, which allows the sender to send multiple packets at once without waiting for an ACK after each one. Pipelining increases efficiency by keeping more packets in transit between the sender and receiver.

Key Changes Due to Pipelining

- Larger Sequence Numbers: Because multiple packets are sent at once, each packet needs a unique sequence number to keep track of its order. The sequence numbers come from a fixed range, like [0, 2^k-1], where "k" is the number of bits used for the sequence numbers.
- Fixed Number of Packets in the Pipeline: The sender can only send a certain number of packets at once, based on the sliding window size (more on this below).
- Transmit and Receive Buffers: The sender and receiver need buffers to store packets temporarily. The sender buffers packets before sending them, and the receiver may buffer packets if they arrive out of order.
- Careful Use of Timers: Timers are set for retransmitting packets that might have been lost or corrupted. The way timers are used is different in each protocol.

Two Types of Pipelining Protocols

There are two common pipelining protocols used for reliable data transfer:

- Go-Back-N (GBN) Protocol
- Selective Repeat (SR) Protocol

Go-Back-N Protocol (GBN)

In the Go-Back-N (GBN) protocol, the sender can send up to N packets without waiting for an acknowledgment. Here's how it works step by step:

1. Sender's Sliding Window:
   - The sender has a sliding window that keeps track of up to N unacknowledged packets (packets waiting for an ACK).
   - It sends these N packets one after the other without stopping.
   - The window slides forward as ACKs come in.
1. Single Timer for All Packets:
   - The sender sets one timer for the oldest unacknowledged packet.
   - If this timer expires before the sender gets an ACK, the sender retransmits all packets in the window starting from the oldest one.
1. Cumulative Acknowledgments (ACKs):
   - The receiver sends ACKs for packets it receives in the correct order.
   - If a packet arrives out of order (or is corrupted), the receiver just sends the last ACK it sent. For example, if packet 1 is missing, the receiver will keep sending ACK 0 until packet 1 is correctly received.
   - The sender considers any ACK as a confirmation that all previous packets have been received correctly (cumulative ACK).
1. How Retransmission Works:
   - If the timer for the oldest packet expires, the sender goes back and retransmits that packet and all subsequent ones, even if some of them were already received successfully.
1. Sequence Numbers:
   - The sender and receiver use sequence numbers to keep track of packets. The number of sequence numbers is limited (based on k-bit numbers), so after reaching the max, it wraps around and starts again.
- In short: GBN is simple but can be inefficient because if one packet is lost, the sender has to retransmit all the packets in the sliding window, even if some of them were received correctly.

Selective Repeat (SR) Protocol

In the Selective Repeat (SR) protocol, the sender can send up to N packets just like in GBN, but the key difference is how it handles packet loss and retransmissions:

- Individual Timers for Each Packet:
  - Instead of one timer for all packets, the sender sets a separate timer for each packet.
  - If a timer for a specific packet expires, only that packet is retransmitted, not all packets in the window.
- Out-of-Order Packets Are Stored:

    - The receiver can accept and store packets out of order.
    - If a packet arrives out of sequence (e.g., packet 3 comes before packet 2), the receiver keeps it and waits for the missing packet(s) to fill in the gaps.
    - Once the missing packet arrives, the receiver processes everything in order.
- Acknowledgments (ACKs):
    - The receiver sends an ACK for each packet it gets, whether it's in order or out of order.
    - If a packet is corrupted, the receiver sends an ACK for the last correctly received packet.
- Sliding Window:
    - The sender's window slides forward as ACKs come in for each individual packet.
    - The receiver also maintains a sliding window to track which packets have been received and stored.
    - In short: SR is more efficient than GBN because it retransmits only the missing or corrupted packets, not all packets in the window. However, SR is more complex because it requires more memory and processing to manage the individual packets and timers.

Key Features of Pipelining Protocols

- Sequence Numbers: Used to identify each packet uniquely and to distinguish between old and new packets.
- Checksums: Used to detect errors in packets.
- Sliding Windows: Both the sender and receiver use sliding windows to keep track of which packets have been sent and which ones need to be acknowledged.
- Timers: Used to retransmit packets if an ACK is not received in time. Choosing a timer value is important:
    - A small timer value causes frequent retransmissions (even when packets are just delayed).
    - A large timer value reduces retransmissions but might lower overall throughput.

Limitations of GBN and SR

Despite their benefits, GBN and SR have several limitations:

- Fixed Timer Values: Both protocols use fixed timer values, which might not adapt well to changing network conditions.
- Fixed Transmission Rate: The size of the sliding window is fixed, so they can't adjust the transmission rate based on how busy or idle the network is.
- Message Segmentation: Breaking messages into packets happens arbitrarily, which might not always be optimal.
- Buffer Overflow: If the receiver can't keep up with the packets being sent, its buffer may overflow, causing packet loss.

- No Two-Way Communication: These protocols focus on one-way communication, meaning only one process can send data at a time. This is inefficient for real-world applications where two-way communication is needed.

TCP: A Smarter Protocol

To address the limitations of GBN and SR, we use Transmission Control Protocol (TCP). TCP is a real-world protocol that:

- Adapts to network conditions by adjusting the timer values and transmission rate based on congestion.
- Supports two-way communication, allowing both the sender and receiver to exchange data at the same time.
- Implements flow control and congestion control to prevent buffer overflow and manage network traffic efficiently.

In summary, TCP is a more advanced and adaptive protocol that overcomes the limitations of GBN and SR, making it the backbone of reliable data transfer on the internet

TCP Segment Structure

- Sequence Numbers:
  - When you send a large message, it gets split into smaller parts, called segments.
  - Each segment has a sequence number, which is the number assigned to the first byte of that segment. These numbers help the receiver know the order of the segments.
  - For example, if a segment has a sequence number of 1000 and contains 500 bytes, the next segment will have a sequence number of 1500.
  - These sequence numbers are chosen randomly when a TCP connection is set up.
- Acknowledgment Numbers:
  - The receiver sends an acknowledgment number (ACK) back to the sender to confirm which data it has received.
  - The acknowledgment number is the sequence number of the next byte the receiver expects. For example, if the receiver sends an ACK number of 1500, it means it has successfully received all bytes up to 1499.
  - If a segment is missing, the receiver stores any out-of-order segments and sends the ACK for the last correct segment. This way, the sender knows to retransmit the missing segment.
  - TCP uses cumulative acknowledgments, meaning that when the receiver sends an ACK for a segment, it assumes that all earlier segments were received.
- Header Length:
  - The TCP segment has a header that contains information about the segment, like sequence numbers, ACKs, and other control data.

  - The header length is usually 20 bytes, but it can extend up to 40 bytes if extra options are included.
  - The options field in the header is used to negotiate settings like the Maximum Segment Size (MSS) and other parameters when establishing a connection.
- Flags:
  - TCP uses several flags (like SYN, ACK, FIN) in the header to control the connection and manage data transfer.
  - SYN: Used to initiate a connection.
  - ACK: Acknowledges the receipt of data.
  - FIN: Used to close a connection.
  - RST: Resets the connection in case of an error.
  - URG: Indicates that urgent data is being sent.

Flow Control and the Receive Window

TCP uses a receive window to prevent overwhelming the receiver with too much data at once. Here's how it works:

- The receive window tells the sender how much space is available in the receiver's buffer.
- The sender can only send as much data as the receiver can handle, ensuring that the receiver's buffer doesn't overflow.
- The receive window size is a 16-bit field that indicates how many bytes of data the receiver can still accept.

Congestion Control

TCP also has mechanisms to manage network congestion and ensure smooth data transfer, even when the network is busy:

- Congestion Window: TCP uses a sliding window called the congestion window to control how many segments are sent at a time. The size of this window changes dynamically based on network conditions.
- CWR and ECE Flags: These flags are used for Explicit Congestion Notification (ECN), which allows the sender and receiver to manage congestion before packets are lost.

TCP Connection Setup: The 3-Way Handshake

Before data can be transferred, the sender (client) and receiver (server) need to establish a connection. This process is called the 3-way handshake, and it happens in three steps:

- SYN: The client sends a SYN (synchronize) segment to the server to start the connection.
- SYN-ACK: The server responds with a SYN-ACK, acknowledging the client's request and agreeing to establish a connection.
- ACK: The client replies with an ACK (acknowledgment), and the connection is now established.

No actual data is transferred during this handshake; it's just for setting up the connection.

Pipelining, Timers, and Cumulative Acknowledgments in TCP

TCP uses concepts from Go-Back-N (GBN) and Selective Repeat (SR) to manage the flow of data:

- Pipelining: Multiple segments can be sent before waiting for an acknowledgment, which speeds up data transfer.
- Timers: Timers are used to manage retransmissions. If a segment's acknowledgment is not received within the timeout interval, the segment is retransmitted.
- Cumulative Acknowledgments: Like GBN, TCP uses cumulative ACKs, meaning an ACK confirms that all previous data was received.

Retransmission and Network Congestion in TCP

One of the most important features of TCP is its ability to adapt to network conditions, especially when the network is congested:

- TCP retransmits lost or corrupted segments but adjusts its retransmission strategy based on network congestion.
- TCP uses a dynamic timeout interval to decide when to retransmit a segment.

Timeout Interval Estimation in TCP

To decide how long to wait for an acknowledgment (before retransmitting a segment), TCP uses an adaptive timeout mechanism:

- Round Trip Time (RTT):
  - RTT is the time it takes for a segment to travel from the sender to the receiver and for the ACK to come back.
  - TCP measures the Sample RTT for a segment and uses it to calculate the Estimated RTT, which is a smoothed average over multiple RTTs.
  - Exponential Weighted Moving Average (EWMA):
    - The Estimated RTT is updated using an exponential weighted moving average formula. This method ensures that recent RTT measurements are given more weight but still considers older measurements.
  - The timeout interval is then calculated based on both the Estimated RTT and its standard deviation (DevRTT).
- Updating the Timeout Interval:
  - If the sender receives all ACKs on time, it calculates a new timeout interval based on the RTT values.
  - If a timeout occurs (i.e., an ACK is missing), the timeout interval is doubled, and the sender retransmits the unacknowledged segments.
- Equation for Timeout Interval:
  - TimeoutInterval = EstimatedRTT + 4 * DevRTT

- Here, EstimatedRTT is the mean RTT, and DevRTT is the deviation (or "jitter") of the RTT.
- TCP uses constants $\alpha = 0.125$ and $\beta = 0.25$ to update the EstimatedRTT and DevRTT values.

Summary of Key Features of TCP

- Sequence Numbers: Used to keep track of the order of segments.
- Acknowledgment Numbers: Used to confirm receipt of data and manage missing segments.
- Sliding Window: Controls how much data can be sent without overwhelming the receiver.
- Flow Control: Uses the receive window to prevent buffer overflow at the receiver.
- Congestion Control: Adjusts the rate of data transfer based on network conditions.
- Timeout Interval: Dynamically adjusted based on network conditions to avoid unnecessary retransmissions and delays.
- TCP's adaptive approach to timers, congestion control, and flow control makes it a reliable and efficient protocol for transferring data over the internet.

TCP Transmissions and Flow Control: Detailed Simple Explanation

TCP (Transmission Control Protocol) is a reliable protocol for transferring data across networks. Here, we'll explain how segments are sent, how acknowledgments work, and how TCP avoids network congestion and prevents data overflow using flow control.

Sequence Numbers and Congestion Window

- Sequence Numbers: When data is sent from one device (host A) to another (host B), it is divided into small chunks called segments. Each segment is assigned a unique sequence number.
- The sequence number helps the receiver reassemble the data in the correct order.
- Congestion Window: The congestion window is the batch of segments that the sender is allowed to transmit before receiving acknowledgments from the receiver. This window helps manage how much data is in transit.
- Example:
  - Let's say we have a series of segments with sequence numbers starting from 92.
  - First segment: Sequence number = 92
  - Next segment: Sequence number = 92 + size of the previous segment (e.g., 100 bytes) = 192

Acknowledgments and Lost Acknowledgments

- Acknowledgments (ACKs): When the receiver gets a segment, it sends back an acknowledgment number (ACK), indicating the sequence number of the next byte it expects to receive. This helps the sender know the data arrived correctly.

- Cumulative Acknowledgment: The receiver can send a cumulative acknowledgment, which means that it acknowledges receiving everything up to the given sequence number.
- Handling Lost Acknowledgments:
  - If an acknowledgment is lost (e.g., for sequence number 92), the sender will retransmit the segment after a timeout.
  - The timeout interval is doubled after each timeout (this is called exponential backoff), to account for potential network congestion.

Cumulative Acknowledgment and Retransmission

- Sometimes, an acknowledgment for a later segment might arrive before the lost acknowledgment.
- Example: Suppose the acknowledgment for sequence number 100 arrives but the acknowledgment for 92 was lost. The sender will not retransmit the segment with sequence number 100 because the acknowledgment is cumulative (i.e., it means all segments before 100, including 92, were successfully received).

Network Congestion and Fast Retransmit

- Congestion Control: Errors like lost segments or delayed acknowledgments happen because of network congestion. TCP adjusts the congestion window size based on network conditions, helping avoid overwhelming the network.

Duplicate Acknowledgments:

- Sometimes segments arrive out of order. When this happens, the receiver waits (e.g., 500 milliseconds) before sending an acknowledgment.
- If out-of-order segments continue arriving, the receiver sends duplicate ACKs for the last correctly received segment.
- If the sender gets three duplicate ACKs, it assumes that a segment was lost and initiates fast retransmit.
- Fast Retransmit: The sender retransmits the missing segment before the timer expires, speeding up recovery.

TCP Flow Control and Receive Window

Buffer Management:

- Each side of the connection allocates a buffer to store incoming segments before they are processed by the application.
- If the application is slow to read data from the buffer, new incoming segments might cause the buffer to overflow.

Receive Window (rwnd):

- The receiver tells the sender how much space is left in the buffer using the receive window (rwnd) field in the TCP header.
- Condition to prevent overflow: The sender should not send more data than the receiver's buffer can handle. This is done by maintaining the following:

  - LastByteSent: The sequence number of the last byte sent by the sender.
  - LastByteAcked: The sequence number of the last acknowledged byte by the receiver.
  - The difference between LastByteSent and LastByteAcked gives the number of segments that are still in transit.
- Example:
  - Receive window (rwnd): The amount of space available in the buffer at the receiver (host B).
  - LastByteRead: The sequence number of the last byte that was read from the buffer by the application at host B.
  - LastByteRcvd: The sequence number of the last byte that arrived and was placed in the receive buffer at host B.
  - To avoid buffer overflow, the sender at host A must ensure that the number of bytes in transit is less than or equal to the size of the receive window (rwnd).

Full-Duplex Communication and Buffer Flow

- Full-Duplex:
  - TCP supports full-duplex communication, meaning both hosts (A and B) can send and receive data simultaneously.
  - Receive Window Updates: Each host informs the other about its buffer status using the receive window field in the ACKs it sends.
  - If host B is receiving data faster than its application can process, its receive window will shrink, and it will notify host A to slow down its sending rate.

Summary:

- TCP Segments: Data is broken into segments, each with a sequence number.
- Acknowledgments (ACKs): Confirm receipt of segments, using cumulative acknowledgments to simplify.
- Lost Acknowledgments: If an ACK is lost, TCP uses timeouts and retransmissions. Fast retransmit helps speed up recovery.
- Flow Control: TCP uses the receive window (rwnd) to manage how much data can be sent without overwhelming the receiver's buffer.
- Congestion Control: TCP adapts the congestion window size and retransmission timers based on network conditions.
- TCP's combination of sequence numbers, acknowledgments, and flow control ensures that data is delivered reliably and efficiently, even in unpredictable network conditions

Congestion Control

- End to End
  - The end system must infer a congestion event (where congestion occurs is irrelevant).

    - The end system reduces the rate of transmission by decreasing the sliding window
    - TCP uses this approach where congestion is inferred from timeout events or triple duplicate ACK events
- Network Assisted
    - A router explicitly sends a notification packet to the sender about the congestion (direct)
    - Alternatively, the router sends a choke packet to the receiver and the receiver in turn notifies the sender (indirect)
- Transmission rate depends on congestion window and RTT
- Congestion window in TCP is adaptive (i.e., it increases and decreases based on network congestion after each round)
- Congestion window (aka sliding window and denoted by cwnd) was fixed arbitrarily under both GBN and SR
    - cwnd is expressed in bytes (LastByteSent – LastByteAcked)
- TCP segment length is called maximum segment size (MSS)
    - Note that MSS is negotiated by sender and receiver using SYN and SYNACK segments via Options field in the TCP header
    - Packet length (L) transmitted = MSS + IP header + link layer header
- Congestion is detected by a sender based on two events
    - Timer expires and some acknowledgements were not received
    - Triple duplicate ACKs were received
- Congestion control algorithm modifies the congestion window size based on the presence or absence of packet loss
    - When no packet losses are detected, then the sender increases the value of cwnd (i.e., sender's rate increases)
    - When packet losses are detected, then the sender decreases the value of cwnd (i.e., sender's rate decreases)
- Classical TCP uses a conservative strategy in exploring network congestion (i.e., probing available bandwidth)

Phases

Congestion control algorithm has 3 phases

- Slow start
    - Initial cwnd is 1 MSS
    - For every new acknowledgement received in the transmission round, cwnd increases by 1 MSS
    - In other words, when all packets in a given transmission round are acknowledged, then the cwnd value of the next round is double the value of cwnd of the current round (i.e., binary exponential increase)

  - When packet loss is detected due to timeout, a slow start threshold (ssthresh) is first calculated and then slow start phase repeats (i.e., go to step 1)
  - During the slow start phase following the timeout event, when cwnd equals ssthresh then transition into congestion avoidance phase takes place
  - Suppose a triple duplicate ACK event is detected during any transmission round, then TCP Reno implements fast recovery phase after the following calculation
- Congestion avoidance
  - The congestion avoidance (CA) phase is initiated when the cwnd value under slow start phase becomes greater than equal to the ssthresh value
  - During the CA phase, when the cwnd is incremented by 1/cwnd for every new acknowledgement received
    - In other words, when all transmitted packets in the given round are acknowledged the cwnd value is incremented by 1
  - A next segment (if waiting in the buffer) is transmitted after receiving a new acknowledgement
  - When timeout event is detected, a new value of ssthresh is calculated (i.e., 0.5×cwnd) and slow start phase commences
  - When a triple duplicate ACK event is detected during the congestion avoidance phase, TCP Reno implements the fast recovery phase after the following calculation
  - ssthresh=0.5×cwnd and then cwnd=ssthresh+3×MSS
- Fast recovery
  - When a triple duplicate ACK event is detected, the timer is terminated immediately and the oldest unacknowledged packet is transmitted immediately
  - Refer to slide 188 (fast retransmit strategy)
  - Then the acknowledgement for this retransmitted packet is awaited by the sender
  - Suppose the transmission is successful, the algorithm transitions to congestion avoidance phase
  - Suppose triple duplicate ACK event is again detected, the algorithm commences slow start phase after the following calculation and retransmits the missing segment

Internet Protocol

Network layer Functions

The transport layer passes the segments to network layer.

The network layer prepares datagrams by inserting IP header

- In other words, segments are encapsulated into datagrams

- The IP headers are sufficient for the routers to guide the datagrams until they reach the destination host
- Functions at host level
  - Obtaining IP address for host
  - Converts segments to datagrams with the specified IP version
  - Error detection of IP header
- Functions at network level
  - Assigning address to hosts
  - Forwarding datagrams
  - Switching operation within a router
  - Routing in the internet

IPv4

Datagram Format

- Version: 4-bit number specifying IP version 4. Routers identify the datagram type using this field.
- Header length: 4-bit field to give the size of the IP header. It is expressed as 32-bit words. When options field is (rarely) present the length exceeds 20 bytes.
- Type of service: 8-bit field for providing differentiated services (e.g., real-time and non-real-time) in networks
- Datagram length: 16-bit field for proving the length of the IP header plus payload.
- Identifier, flags, fragmentation offset: IPv4 routers fragment large datagrams based on the MTU supported by the links reassembling the original datagram at the destination host. Identifier is a 16-bit field, flags occupy 3 bits and fragmentation offset is a 13-bit field
- Time-to-live (TTL): 8-bit field which indicates the number of hops the packet can travel without being discarded. A router decrements this field by 1 upon processing
- Protocol: 8 bit field to describe the upper layer protocol for the datagram (e.g., 6-TCP, 17-UDP, 1-ICMP)
- Header checksum: 16-bit field for error detection. Only the header fields are used in the calculation. Whenever a datagram is forwarded or fragmented this field is updated
- Source and destination IP addresses: 32-bit fields each. Routers use the destination IP field for delivering the datagrams to the destination host. Give a usage of source IP address
- Options: This field is rarely used and has been removed in IPv6.
- Payload: Variable length field which holds the upper layer information encapsulated into the datagram. The max size of payload is 65,535 bytes, however, rarely this size occurs. Datagram size in Ethernet range is 46-1500 bytes

Addressing

1. todo

- The network address is of the form a.b.c.d/x where a, b, c and d are 8 bit decimal numbers and x is prefix
- For example, 192.168.1.0/24 means that the first 24 bits are common among the IP addresses in the subnet. Remaining 32–x bits are variable i.e., all zeros to all ones of length 32–x
- The above format is called Classless Interdomain Routing (CIDR)
- Subnet mask is an alternate way of representing the prefix in 32-bit dotted decimal format.
- The subnet mask is expressed as x ones and 32–x zeros

DHCP

- Dynamic host configuration protocol
  - Client server model for obtaining IP address
  - Server listens for UDP messages on port 67
- DHCP allows a host to obtain (be allocated) an IP address automatically
- Host communicates initially using broadcast address
  - Broadcast address: 255.255.255.255
- Host sends a DHCP discovery message
- DHCP servers replies with DHCP offer messages
- Host replies with DHCP request
- DHCP server acknowledges with the assigned IP address

NAT

- Network address translation
- IP addresses allotted by an ISP to a consumers (e.g., home networks, enterprise networks) could be insufficient
- In such cases, private IP networks are created and the public IPs, allotted by the ISPs, are shared
- Private IP address can be assigned manually or using DHCP
- Private hosts can access the public internet using NAT
- NAT maps the private IP addresses to the public IP addresses
- NAT table maintains such mapping
- Interfaces of NAT-enabled routers have some interfaces assigned to private IP addresses whereas some in interfaces assigned the public IP addresses given by the ISP

IPv6

- Proposed by IETF (actually started to address the limitations of IPv4)
- Initiated in 1998 and came into effect on 6 June 2012
- Provides 3.4 ×1038 IP addresses of 128 bits each
- Currently implemented by new ISPs and carriers
- About 22% of the IP traffic is due to IPv6 as of 2018
- Fixed 40 byte header compared to the 20 byte (variable length) IPv4 header

- It is more robust to IP spoofing and attacks
- Besides unicast and multicast (IPv4 modes), supports anycast communications
- Key differences with IPv4
  - Doesn't have:
    - options, header length, internet checksum and datagram fragmentation
  - Has:
    - 128 bit IP addresses, flow label and payload length

Datagram Format

- Version: 4-bit field identifies the IP version number.
- Traffic class: 8-bit traffic class field, like the TOS field in IPv4. Used to give priority to certain datagrams within a flow, or it can be used to give priority to datagrams from certain applications
- Flow label: 20-bit field is used to identify a flow of datagrams
- Payload length: 16-bit value is treated as an unsigned integer giving the number of bytes in the IPv6 datagram following the fixed-length, 40-byte datagram header.